School year 2017-2018

Software engineering's project

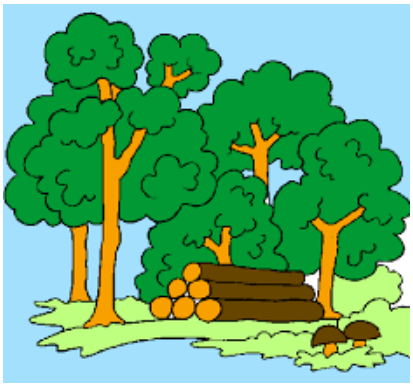
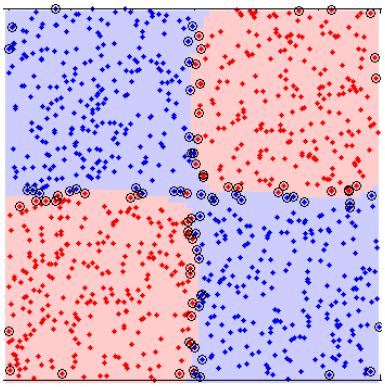
*

Web project

Final report

Back2Back testing

ML algorithms implementation

Supervisors

Acher Mathieu

Barais Olivier

Boursier Johann

Students

Coray Yoann

Dubouloz Pierre

Humbert Lucas

Lesoil Luc

Introduction

Objectif

Le but de ce projet est, à partir de java et de ses nombreuses librairies de machine learning, de comparer l'accuracy des modèles calculés en fonction des librairies choisies, des algorithmes choisis et de quelques autres paramètres.

Dans notre projet, nous nous sommes concentrés sur trois librairies différentes : Renjin, Weka et SparkML.

Dans chacune de ces librairies nous avons implémenté trois algorithmes de machine learning différents : l'arbre de classification, la forêt aléatoire et le Support Vector Machine (SVM).

Structure du projet

Notre code se lance avec la *main* de la classe Main. Elle crée une console afin d'interagir avec l'utilisateur. Celui-ci renseigne les paramètres nécessaires aux calculs du modèle et de son accuracy. Grâce à des expressions régulières, l'utilisateur est prévenu de toute erreur de saisie lors de cette interaction avec la console.

La généralisation de notre projet nous a amené à créer une classe abstraite Library de laquelle vont hériter nos trois classes représentant nos trois librairies. Ainsi lorsque nous lançons le programme, nous créons une instance Library et selon ce que l'utilisateur souhaite, ce sera une instance de Renjin, Weka ou SparkML.

Pour chacune de ces librairies, nous avons inclus trois méthodes de machine learning. Nous avons implémenté ces méthodes de la même manière que précédemment : une classe abstraite représentant une méthode. Nous créons ensuite une instance de la classe fille que l'utilisateur souhaite. Par exemple, pour la librairie Rengine : l'arbre de classification CART, "Classification And Regression Trees" est représenté par la classe RengineCART, la méthode random forest par la classe RengineRandomForest, et la méthode de SVM (Support Vector Machine) par la classe RengineSVM.

Diagramme Uml

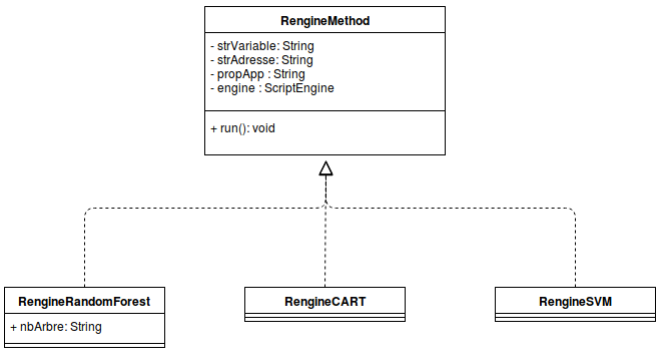


Diagramme UML du package Rengine.

Les deux autres packages (Weka et SparkML) suivent le même schéma.

Maintenant, nous allons expliquer plus en détail l'implémentation des classes et les librairies utilisées dans ce projet que sont Renjin, Weka et SparkML.

Renjin

Explications

Renjin est une librairie permettant d'utiliser du code R avec java. Pour cela, on instancie une Renjinfactory (design pattern factory) qui crée un script renjin. Ce script renjin (ScriptEngine) permet d'évaluer le code R à l'aide de la méthode eval().

Les classes codant Renjin sont lancées (dans le package rengine) depuis la classe Rengine.java. Chaque algorithme est implémenté suivant le modèle de la classe abstraite RengineModel.java.

Chaque méthode fait appel à un script R différent (cart.R pour cart, randomForest.R pour la forêt aléatoire, svm.R pour le svm). Le CART est codé avec rpart, le svm avec e1071, et le random forest avec le package randomForest.

On lit les script R ligne à ligne, en utilisant un Buffer. Et grâce au ScriptEngine créé, et la méthode eval, nous pouvons exécuter les lignes une à une, comme avec un code R. La dernière ligne rendue est donc l'accuracy recherchée.

Les classes héritant de la classe RengineMethod étant très proches nous aurions peut être pu tout regrouper en une seule classe. Elle prendrait juste le fichier R correspondant à l'algorithme souhaité par l'utilisateur.

Test des modèles

A = 70%	Cart	Random Forest (500 trees)	SVM
Iris	0,99	0,99	1
Pages	0,94	0,93	0,9

Accuracy moyenne du modèle en fonction de la base de données et de l'algorithme utilisé. La moyenne est réalisée sur dix modèles.

Weka

Explications

Weka utilise des fichiers Attribute-Relation File Format (.arff), on doit donc créer dans un premier temps une classe CSV2Arff qui convertit les .csv en fichiers .arff. On utilise pour cela un design pattern singleton, pour éviter d'instancier trop d'objets.

Comme précédemment, le package Weka possède une classe abstraite WekaMethod à partir de laquelle sera construite les différentes classes implémentant les algorithmes de machine learning.

Pour la méthode CART, nous utilisons la classe J48, la classe SMO pour le SVM et RandomForest pour la forêt aléatoire.

Ensuite, nous créons une Instance, représentant les données. Après division de l'échantillon pour l'apprentissage et le test, nous pouvons facilement calculer le modèle puis calculer son accuracy.

Concernant la partie arbre de décision, nous créons un png qui va nous permettre de rendre graphiquement l'arbre que nous donne le modèle calculé (dans le src, voir pages.png et iris.png). Un de nos objectifs est d'afficher ces images dans la page web des résultats du projet web.

Test des modèles

A = 70%	Cart	Random Forest (500 trees)	SVM
Iris	0,87	0,99	0,99
Pages	0,86	0,94	0,96

Accuracy moyenne du modèle en fonction de la base de données et de l'algorithme utilisé. La moyenne est réalisée sur dix modèles.

SparkML

Explications

SparkML a été une librairie assez difficile à mettre en place. Nous avons commencé par créer une classe SparkConnection qui va nous permettre de nous connecter à une session Spark. De là, nous avons mis en forme les données (en gérant les types des variables) et avons créé un dataset sur lequel nous allons pouvoir travailler.

Le travail sera, comme pour les autres librairies, laissé aux classes filles de la classe abstraite SparkMLMethod qui implémentent chacune un algorithme de machine learning différent. Nous pouvons ensuite calculer l'accuracy des modèles selon l'algorithme utilisé pour le calculer.

Nous avons rencontré plusieurs difficultés pour implémenter cette partie de notre projet.

Premièrement, la majeure partie de la documentation trouvée sur internet ne concerne que le langage de programmation scala, et pas java. En effet, Spark est plutôt connu pour sa synergie avec scala. Cela a rendu la résolution de certains problèmes plutôt compliquée.

Deuxièmement, nous avons eu un problème de serialization avec certaines méthodes du package SparkML lorsque nous avons voulu généraliser notre code. C'est pour cela que notre classe Main implémente l'interface Serializable et que nous avons du créer une classe RunnableSparkML en plus de la classe SparkML.

Dans cette partie, il y a un léger problème, les seules méthodes permettant de calculer un modèle grâce à un Support Vector Machine que nous avons trouvées ne fonctionnent que si la variable à expliquer est binaire. Ici, seules les données sur les pages permettent au SVM de calculer un modèle et donc calculer son accuracy (cela ne fonctionnera pas sur les iris).

Test des modèles

A = 70%	Cart	Random Forest (500 trees)	SVM
Iris	0,97	0,94	
Pages	0,96	0,93	0,91

Accuracy moyenne du modèle en fonction de la base de données et de l'algorithme utilisé. La moyenne est réalisée sur dix modèles.

Comparaisons des librairies et des modèles

Nous allons maintenant comparer un peu les librairies et leurs algorithmes de machine learning.

En regardant les trois tableaux ci-dessus, nous pouvons remarquer que :

— l'algorithme CART calculé par la librairie Weka nous donne des accuracy plus faibles que pour les autres librairies,

— Renjin semble performant sur des données du type des iris (un exemple de classification réputé en statistique pour ses bons résultats),

— pour la forêt aléatoire, SparkML semble être une librairie moins performante.

Meilleure librairie	CART	Random Forest	SVM
Iris	Rengine	Rengine & Weka	Rengine
Pages	SparkML	Weka	Weka

Meilleures librairies par méthode et par jeu de données.
La comparaison est basée sur l'accuracy.

Lancement du projet

Pour lancer le projet, il suffit d'appuyer sur le bouton run de la class Main (du package Main). Des informations vous seront alors demandées via la console. Il faudra alors renseigner le chemin du fichier csv, le nom de la variable à expliquer, le taux de division de l'échantillon entre l'apprentissage et le test. Il vous sera aussi demandé de choisir la librairie (Weka, SparkML ou Renjin) et l'algorithme à utiliser (arbre de classification, forêt aléatoire ou SVM). Dans le cas de la forêt aléatoire, le nombre d'arbres vous sera demandé.

Attention, il existe certaines contraintes :

— les variables du fichier csv doivent toutes être explicatives, à l'exception de la première (identifiant),

— le nom des variables ne doivent pas contenir de points,

— le svm de SparkML ne fonctionne que si la variable à expliquer est binaire.

Après calcul et les diverses informations données (différentes selon les librairies et les algorithmes), nous obtenons l'accuracy. Nous pouvons ainsi comparer nos résultats.

Nous avons laissé une *main* dans une classe de chaque package afin de pouvoir lancer plus rapidement un algorithme sur la librairie souhaitée. Voici les classes que nous pouvons lancer : Rengine, WekaMain et SparkML.